Received May 31, 2017, accepted June 14, 2017, date of publication July 7, 2017, date of current version July 24, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2723758

PR-LRU: A Novel Buffer Replacement Algorithm Based on the Probability of Reference for Flash Memory

YOUWEI YUAN[1,2], YETING SHEN[1], WANQING LI[1], DONGJIN YU[1], (Member, IEEE), LAMEI YAN[1], AND YIFEI WANG[1]
[1]School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China
[2]Key Laboratory of Complex Systems Modeling and Simulation, Ministry of Education, Hangzhou 310018, China

Corresponding author: Youwei Yuan (yyw@hdu.edu.cn)

This work was supported in part by the Natural Science Foundation of Zhejiang Province under Grant Y17E050119 and in part by the Research Center of Information Technology & Economic and Social Development in Zhejiang Province under Grant 15XXHJD04.

ABSTRACT NAND flash memory has many advantages, including a small form factor, non-volatility, and high reliability. However, problems caused by physical limitations, such as asymmetric I/O latencies and out-of-place updates, still need to be resolved. By using a probability of reference (PR) to select a candidate page as the victim page, this paper presents a novel buffer replacement algorithm called PR least recently used to enhance the flash memory performance. To predict whether a page may be referenced in the future, three variables are used to calculate a page's PR. In addition, we improve the performance overhead of the number of write operations, the hit ratio, and the runtime using a novel PR strategy. The algorithm is implemented and tested on the flash simulation platform Flash-DBSim. The results indicate that our algorithm provides improvements of up to 7% for the hit ratio with an improvement of up to 36.7% for the overall runtime compared with other approaches.

INDEX TERMS Buffer replacement algorithm, flash memory, flash-based DBMS, probability of reference.

I. INTRODUCTION

During the past decades, flash memory has been used extensively because of its high reliability, small size, light weight, shock resistance, and power economy. Due to its advantageous features and decreasing price, NAND flash memory has been used to save the setting information in various devices, including in computers, personal digital assistants (PDA), and the BIOS of digital cameras. Flash memory does not lose the data when the device is powered down which has been used in the system's memory hierarchy [1]–[3]. Based on the advantages of flash memory, solid state disk (SSD) has become a popular choice in an enterprise computing environment. Flash memory is a type of bulk erase and out-of-place update media [4]–[7]. These characteristics provide improvements for the buffer replacement algorithm in the flash memory [8], [9].

Traditional buffer replacement algorithms are designed for disk and are not suitable for NAND flash memory [10]. These disk-oriented buffer replacement algorithms are based on the same latency as for read and write operations. Applying these traditional disk-oriented buffer replacement algorithms directly to flash memory does not result in the advantages of flash memory and is not conducive to the capability of NAND flash memory. When using flash memory, it is very important to redesign the flash-oriented buffer replacement algorithm [11]–[14].

Many current NAND flash-based buffer replacement algorithms have focused on decreasing the number of write operations, improving the hit ratio. These algorithms modify the Least Recently Used (LRU) algorithm [15]. When a page has recently been referenced, LRU has a higher likelihood of being referenced in the future. This is the base of the LRU algorithm. However, these algorithms ignore certain information, such as the reference locality and the reference times of the pages. This information can be used to further improve the flash performance [16].

In this study, a novel flash-based buffer replacement algorithm called PR-LRU (Probability of Reference LRU) has been presented to solve the asymmetric I/O cost. We use a reference probability to predict the possibility that a page is referenced in the future. A page's reference probability is calculated using three variables, namely the reference times, the number of reference pages from the last to the penultimate references, and the number of reference pages from the first to the last reference. Moreover, a victim LRU list provides an additional chance for the page to be stored in the flash memory.

The remainder of this article is structured as follows. The related works in this field is introduced in Section 2 prior to describing the proposed buffer replacement algorithm PR-LRU in Section 3. In section 4, the detailed analysis of the simulation experiment and the results are described using various traces while Section 5 provides the conclusion and outlines our future work.

II. RELATED WORKS

Flash memory has many disadvantages including asymmetric I/O latencies, not-in-place updates, and a slow erase operation. Table 1 shows the latency of DRAM and NAND flash memory in I/O operations. This shows that the flash write operation requires more time than the read operation and that DRAM does not require the erase operation. In addition, due to the physical features of the NAND flash, the number of erase operations is limited to within the range of 10,000 to 1,000,000 [17]–[19].

TABLE 1. The I/O performance for DRAM and NAND flash.

Operation	Access time	
	DRAM	NAND flash
Read(2KB)	12.7ms	20μs
Write(2KB)	13.7ms	200μs
Erase(128KB)	N/A	1.5ms

The erase operation in the flash memory does not consist of a single byte but a fixed block and the write operation must be performed in a blank area. If the target area has already been used, it must be erased before it can be re-written. The read operations require less latency than the write/erase operations. Due to the flash memory's asymmetric I/O latencies, current algorithms have focused on reducing the number of write operations.

Several algorithms have been presented for reducing the number of write operations to enhance the I/O performance. Many algorithms will delay the process for evicting dirty pages from the buffer [20]–[23]. Park *et al.* [20] presented a Clean-First LRU (CF-LRU) algorithm to solve the asymmetric I/O latencies. The main concept of the CF-LRU is replacing the clean pages and retaining the dirty pages as long as possible. Designed by Jung, the LRU-WSR algorithm uses a Write Sequence Reordering (WSR) strategy [21]. It assigns a cold flag to every dirty page to determine if a dirty page is cold or not. Based on the CF-LRU, the Cold-Clean-First LRU (CCF-LRU) algorithm was published by Li *et al.* [22]. This algorithm will evict the cold clean pages first. When buffer is full of hot and dirty page, other pages will be evicted with the help of a cold-detection mechanism. The CCF-LRU algorithm always gives priority to replacing the clean pages, which results in the immediate replacement of the page that was just read in the buffer, thus reducing the hit ratio. Using a modification of the CCF-LRU, Jin et al. introduced the Adaptive Double LRU (AD-LRU) algorithm to further improve the runtime efficiency [23]. The AD-LRU sets the minimum length of the cold queue, so it can dynamically adjust the length of the hot and cold queues.

To guarantee a good flash memory performance during flash I/O operations, previous algorithms were presented to redesign the flash-based buffer replacement algorithm [13], [20]–[23]. Current works research has given priority to the replacement of cold clean pages and does not take the reference times and other information into account [24]–[27]. Therefore, the existing buffer replacement algorithm can still be optimized.

III. THE PROPOSED PR-LRU ALGORITHM

A. THE STRUCTURE DESIGN OF THE PROPOSED PR-LRU ALGORITHM

A novel buffer replacement algorithm for NAND flash called PR-LRU (Probability of Reference LRU) is proposed to increase the buffer hit ratio by using a probability of reference to choose a candidate page as the victim. Fig. 1 illustrates the structure of the presented PR-LRU algorithm, which contains the hot LRU list, the cold LRU list, and the victim LRU list. The length of the hot LRU list is L_1. It maintains the hot pages. Regardless of whether the page is dirty or clean, the cold LRU list contains L_2 pages. The victim LRU list contains L_3 pages and only stores the pages that are replaced from the hot or cold LRU list. Moreover, L_1, L_2 and L_3 add up to the buffer size.

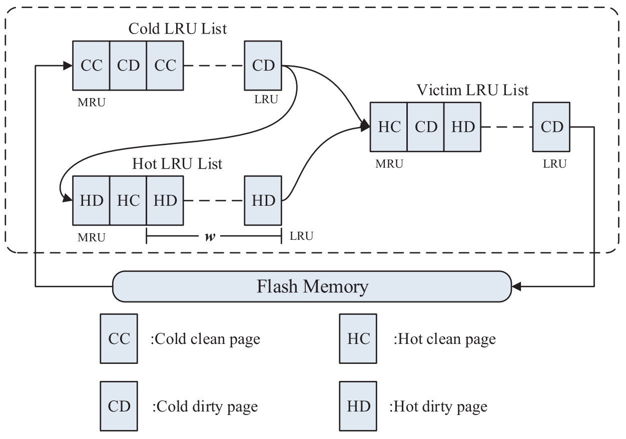

FIGURE 1. The structure of the proposed PR-LRU algorithm.

B. BUFFER REPLACEMENT STRATEGY

Three variables are used to calculate the probability of reference, namely the reference times, the number of reference pages from the last to the penultimate reference, and the number of reference pages from the first to the last reference. A page that is referenced twice or more is called a hot page, otherwise, it is called a cold page. A page that has been re-written is called dirty, otherwise, it is called clean. For the features of flash memory, we designed the algorithm to retain the dirty pages in the buffer if possible. A clean page is preferred as a replacement in the buffer. Moreover, for the sake of overall performance, evicting cold pages are better than evicting hot pages.

The probability of reference will be calculated using the following four Theorems. The locality of current reference page will be considered in Theorem 1. Theorem 2 takes the lifecycle of the reference page into account. Theorem 3 computes the overall average number of references. The value of Theorem 1, Theorem 2 and, Theorem 3 are used to calculate the three parts of the probability of reference in Theorem 4.

Theorem 1: Given that the number of the reference pages is designated as *number* and the number of references from the first reference to the last references is designated as $dist_i$, $1 \le i \le w$. $log_2 i * q$ represents the cost of the write and q is the weight of the write operation. Then, the probability of the average reference page can be calculated using Formula (1):

$$Probability\,(dist_i, number) = log_2 i * q * \left(1 - \frac{1}{1 + e^{-\frac{dist_i}{number}}}\right) \sim (0, 1)\,, \quad 1 \le i \le w. \tag{1}$$

Theorem 2: Let $near_i$ be the last reference to the penultimate reference pages, $1 \le i \le w$. The probability of the recently reference interval pages is given by Formula (2):

$$Probability\,(near_i) = log_2 i * q * \left(1 - \frac{1}{1 + e^{-near_i}}\right) \sim (0, 1)\,, \quad 1 \le i \le w. \tag{2}$$

Theorem 3: Where $\bar{X} = total\_record/number\ of\ pages$ represent the average number of times each page has been referenced. *total_record* represents the sum of reference request. *number* represents how many pages have been referenced in the trace. The probability of reference pages is subsequently computed as Formula (3):

$$Probability\left(number, \bar{X}\right) = \frac{1}{1 + \frac{number}{\bar{X}}} \sim (0, 1)\,, \quad 1 \le i \le w. \tag{3}$$

Theorem 4: Obtain the probability of the average reference pages, the probability of the recently referenced interval pages, and the probability of the reference pages together. The minimum probability of the pages will be selected as the victim, which can be computed using Formula (4):

$$\begin{aligned} Index = \min\{&Probability(dist_i, number) + Probability(near_i) \\ &+ Probability\left(number, \bar{X}\right)\}, \quad 1 \le i \le w. \end{aligned} \tag{4}$$

The probability of reference is used to determine which page to evict. *index* is the page that is selected as the replacement page. The reference probability has been calculated in the *w* pages closest to the hot LRU position. When the algorithm requires a replacement page, the minimum probability page will be selected. In order to reduce the number of unnecessary calculations, a window size *w* is set in the hot LRU list to limit the number of calculated pages.

To sum up, the main strategy of the Probability of Reference (PR) is described as follows:

1. The victim LRU list is used to maintain pages that are evicted from the cold or hot LRU list. Our method gives a second chance to each page to retain the pages in the buffer as long as possible.

2. The probability of reference is used to calculate which pages may be referenced in the future. Through the probability of reference, we estimate whether the page should be removed from the hot LRU list.

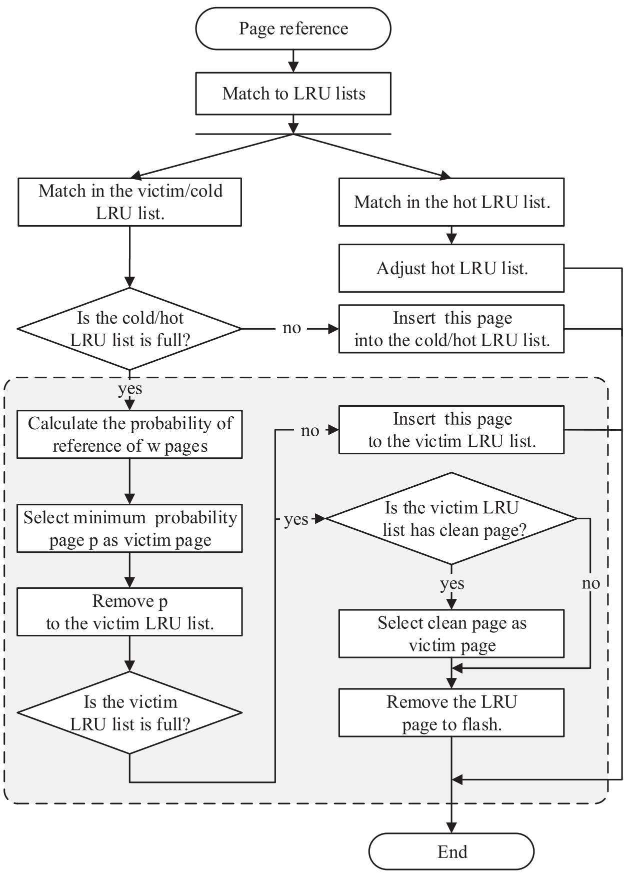
Page reference
Match to LRU lists
Match in the victim/cold LRU list.
Match in the hot LRU list.
Adjust hot LRU list.
Is the cold/hot LRU list is full?
no
Insert this page into the cold/hot LRU list.
yes
Calculate the probability of reference of w pages
no
Insert this page to the victim LRU list.
Select minimum probability page p as victim page
yes
Is the victim LRU list has clean page?
yes
no
Remove p to the victim LRU list.
Select clean page as victim page
Is the victim LRU list is full?
Remove the LRU page to flash.
End

FIGURE 2. Workflow of the PR-LRU.

C. THE WORKFLOW OF THE PR-LRU

Fig. 2 shows the workflow of the PR-LRU. If we receive a page request, the algorithm will determine whether the page is in the buffer. When the requested page is detected in the buffer, this page will be moved to the three LRU lists and will be placed into the MRU location of the hot LRU list. If there is no free space in the hot LRU list, the page with the minimum reference probability in the *w* pages will be selected. When the algorithm need to select a victim page in the hot LRU list, the minimum reference probability in the w pages will be chosen. The method of calculating the probability of reference is described in previous section. This victim LRU list evicts the clean page first unless there is no clean page.

Algorithm 1 PR-LRU

```
Input:   $L_{cold}$: the cold LRU list, $L_{hot}$: the hot LRU list,
         $L_{victim}$: the victim LRU list
Output:  the victim page for replace
1.   if page is in $L_{hot}$ // p is in the hot LRU list
2.     Move page to MRU location of $L_{hot}$
3.   else if page in $L_{cold}$ then // page
     is in the cold LRU list
4.     if there is no free space in $L_{hot}$
5.       victim =SelectVictim () in $L_{hot}$
6.       if there is no free space in $L_{victim}$
7.         victim2 =SelectVictim () in $L_{victim}$
8.         Write victim2 to flash memory
9.       end if
10.        Move victim to MRU location of $L_{victim}$
11.     end if
12.     Move page to the MRU location in $L_{hot}$
13.   else if page in $L_{victim}$ then // page
      is in the victim LRU list
14.     if there is no free space in $L_{hot}$
15.       Move page to MRU location of $L_{hot}$
16.       victim =SelectVictim () in $L_{hot}$
17.       Insert victim to MRU location of $L_{victim}$
18.     else Insert page to MRU location of $L_{hot}$
19.     end if
20.   else // page is not in the buffer
21.     if there is no free space in $L_{cold}$
22.       victim =SelectVictim () in $L_{cold}$
23.       if there is no free space in $L_{victim}$
24.         victim2 =SelectVictim () in $L_{victim}$
25.         Write victim2 to flash memory
26.       end if
27.     Insert victim to MRU location of $L_{victim}$
28.     end if
29.   end if
30.     Insert page to MRU location of $L_{cold}$
31.   return a reference of page
```

D. DETAILED DESIGN OF THE ALGORITHM

Algorithm 1 shows the specific implementation of the PR-LRU. When the required page is detected, this requested page will be put into the MRU position (lines 1-2). If the required page is matched in the cold LRU list and there is no free space in the hot LRU list, the victim page will be selected and placed into the victim LRU list (lines 3-5). When the victim LRU list has no free space, the algorithm 1 will evict a page to the flash memory to make additional space (lines 6-9). The reference page would be put into the MRU position of the hot LRU list (lines 10-18). If the page is found in the victim LRU list, this page will be moved to the hot LRU list. If there is no free space in the hot LRU list, the victim page will be chosen as a replacement (lines 13-19). If the requested page is not found, this requested page will be put into the cold LRU list (lines 20-31). When the cold LRU list has no free space, the page in the LRU position will be selected to move to the victim LRU list (lines 20-29). Then this reference page will be inserted into the cold LRU list (line 30).

Algorithm 2 shows the algorithm SelectVictim of the PR-LRU. If we need to select a victim page in the cold LRU list, the page in the LRU location will be selected directly (lines 1-2). When algorithm 2 selects a victim page from the victim LRU list, it is more prone to choose a clean page for replacement. Dirty pages will be evicted if the victim LRU list is fulled with dirty pages (lines 3-13). When algorithm 2 chooses the victim page in the hot LRU list, the minimum probability page will be selected in the w pages closest to the LRU location (lines 14-16). The method of calculating the probability of reference described previously is used.

Algorithm 2 SelectVictim

```
Input:   A LRU list L
Output:  the victim page for replace
1.    if L is $L_{cold}$
2.      select the LRU page in $L_{cold}$ as victim
3.    else if L is $L_{victim}$
4.      for sit = LRU to MRU
5.        Select the sit page in $L_{cold}$ as q
6.        if (q is clean)
7.          victim = q
8 .         break
9.        end if
10.     end for
11.     if (victim is null)
12.       Select the LRU page in $L_{victim}$ as victim
13.     end if
14.   else
15.     Select minimum probability in
        [LRU, LRU-w] as victim
16.   end if
17.   return a reference to the victim
```

IV. PERFORMANCE EVALUATION

The experiments are implemented using the simulation platform Flash-DBSim. Synthetic traces are used to analyze the hit ratio, runtime, and the write count. The simulation results are summarized and compared with the other three buffer replacement algorithms, namely LRU [13], LRU-WSR [21], and AD-LRU [23] respectively.

A. EXPERIMENT SETUP

Our experiments are implemented using a 3.35 GHz Intel CPU with 16 GB RAM. The operating system is

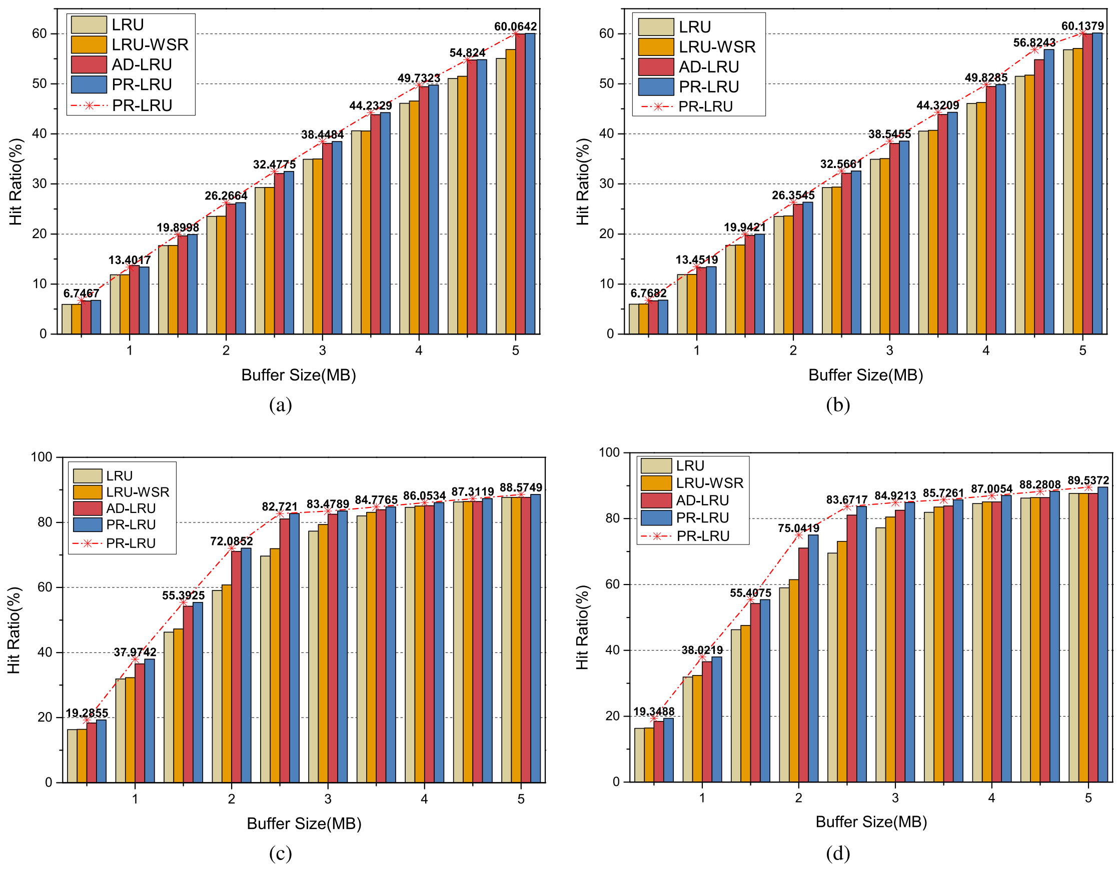

FIGURE 3. Comparison of the hit ratio on different traces. (a) T_1. (b) T_2. (c) T_3. (d) T_4.

TABLE 2. Characteristics of the NAND flash.

Attribute	value
Block size	64pages
Page size	2048B
Write latency	200μs/page
Erase latency	1.5ms/page
Read latency	25μs/page
Endurance	100,000

TABLE 3. Details of the simulated traces.

Type	Total Buffer requests	Memory references		
		Read Operation	Write Operation	Reference Locality
T_1	200,000	18,000(90%)	2,000(10%)	60%/40%
T_2	200,000	14,000(30%)	6,000(70%)	60%/40%
T_3	200,000	10,000(50%)	10,000(50%)	80%/20%
T_4	200,000	16,000(80%)	4,000(20%)	80%/20%

Windows 10 Pro SP1 with 64-bits. The Flash-DBSim is a simulated flash memory platform, which provides a simulation environment for validating the performance of various algorithms. Flash-DBSim is designed for comprehensive experiments and simulates the various experimental conditions that have developed in flash memory technology research. FlashDBSim uses a modular design approach, which includes Virtual Flash Device Module (VFD), Memory Technology Device Module (MTD), and Flash Translation Layer Module (FTL). Researchers can modify the contents of each module according to their actual research needs.

The characteristics of the NAND flash are described in Table 2. We simulated a 128MB NAND flash. The flash data page size is 2KB, and each data block contains 64 data pages. We assumed that write latency is 200μs per page, erase latency is 25μs per page, and read latency is 25μs per page. The number of erase operations is limited to 100,000.

The details of the four types of traces are listed in Table 3. Each trace has 200,000 buffer requests and 10,000 different pages. For instance, 18,000 (90%) indicates that this trace has 2,000 write operations and 18,000 read operations. The reference locality "60%/40%" represents that 60% of the total references are densely performed in 40% of the pages. The size of the page is 2 KB in this experiment.

Three performance metrics, write count, hit ratio, and runtime were used in our simulation experiments to evaluate the results. The erase operations were not considered because they are always conducted prior to the write operations and are equal in number to the write operations. Because the

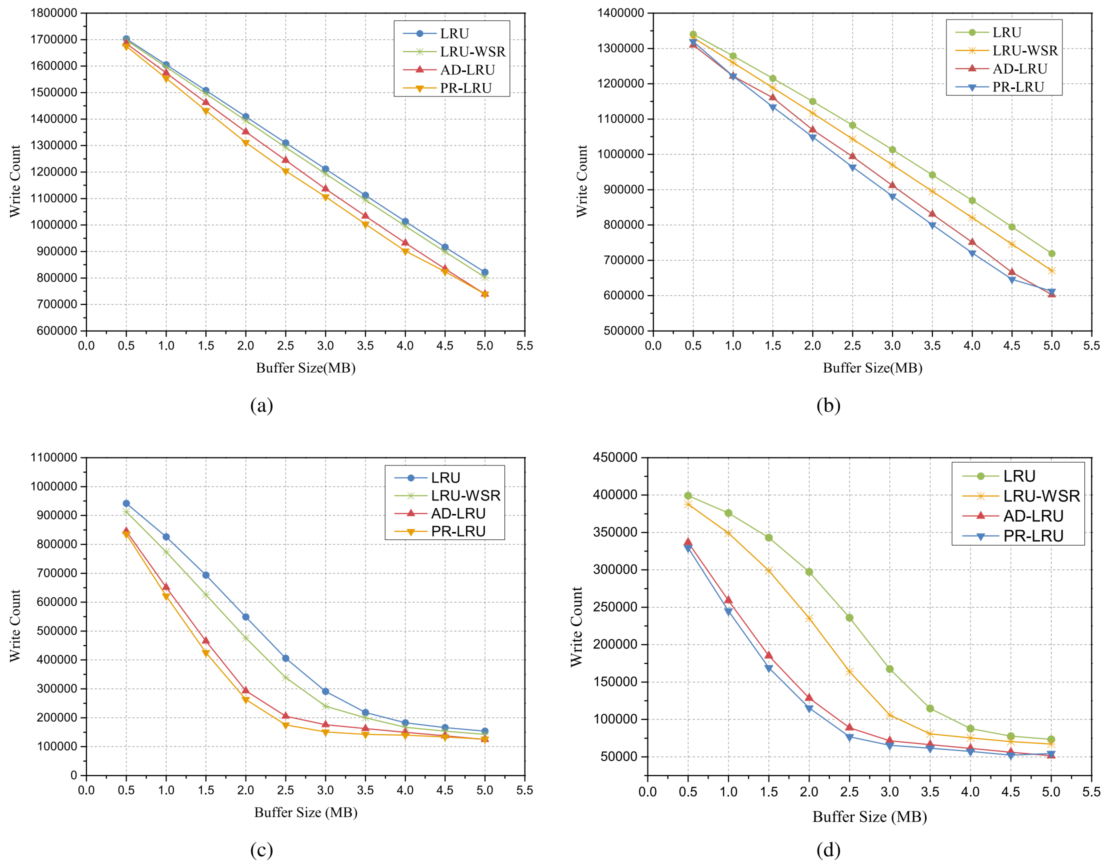

FIGURE 4. Comparison of the write count on different traces. (a) T_1. (b) T_2. (c) T_3. (d) T_4.

TABLE 4. Details of the simulated traces.

Buffer Size(MB)	Hit Ratio (%)					Write Count(10^3)					Runtime (s)				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
LRU	31.87	58.99	77.21	84.55	87.63	375	279	167	97	73	116	85	47	25	21
LRU-WSR	32.36	61.46	80.47	85.08	87.65	348	235	105	75	67	110	70	32	22	19
AD-LRU	36.53	71.07	82.52	85.11	87.66	259	128	71	61	55	88	39	23	20	16
PR-LRU	38.02	75.04	82.42	87.01	89.54	245	115	65	57	54	86	36	20	18	16

latency of the read operation is much faster than that of the write operation in the flash memory, we did not compare the number of read operations with those of other buffer replacement algorithms.

B. PERFORMANCE EVALUATION OF THE SYNTHESIZED TRACES

The experimental results are representative of the results for trace T_4. Therefore, the results of trace T_4 are listed in Table 4, which shows the write count, hit ratio, and overall runtime of the four buffer replacement algorithms. As the memory increases, the trend in the performance improvement gradually stabilizes for the write count, hit ratio, and runtime. It is also evident that the PR-LRU redesign is highly appropriate for the flash memory. An appropriate increase in the buffer size is conducive to the performance of the replacement algorithm. However, simply increasing the buffer size will waste resources.

Fig. 3 illustrates the comparison of the hit ratio for the different traces and for various buffer sizes. Among the four traces, PR-LRU has a better hit ratio than the other algorithms. Because the PR-LRU takes the locality of the pages into account based on the LRU algorithm, it has a higher hit ratio. As a consequence, the increase in the hit ratio for the PR-LRU and trace T_4 compared to LRU [13], LRU-WSR [21] and AD-LRU [23] was 7%, 5%, and 2% respectively. PR-LRU not only improves the performance of the flash memory but it also enhances the performance of the buffer replacement algorithm.

Fig. 4 shows a comparison of the write count for the different traces. When the buffer size is particularly large or small, we observe that the write count is similar for the

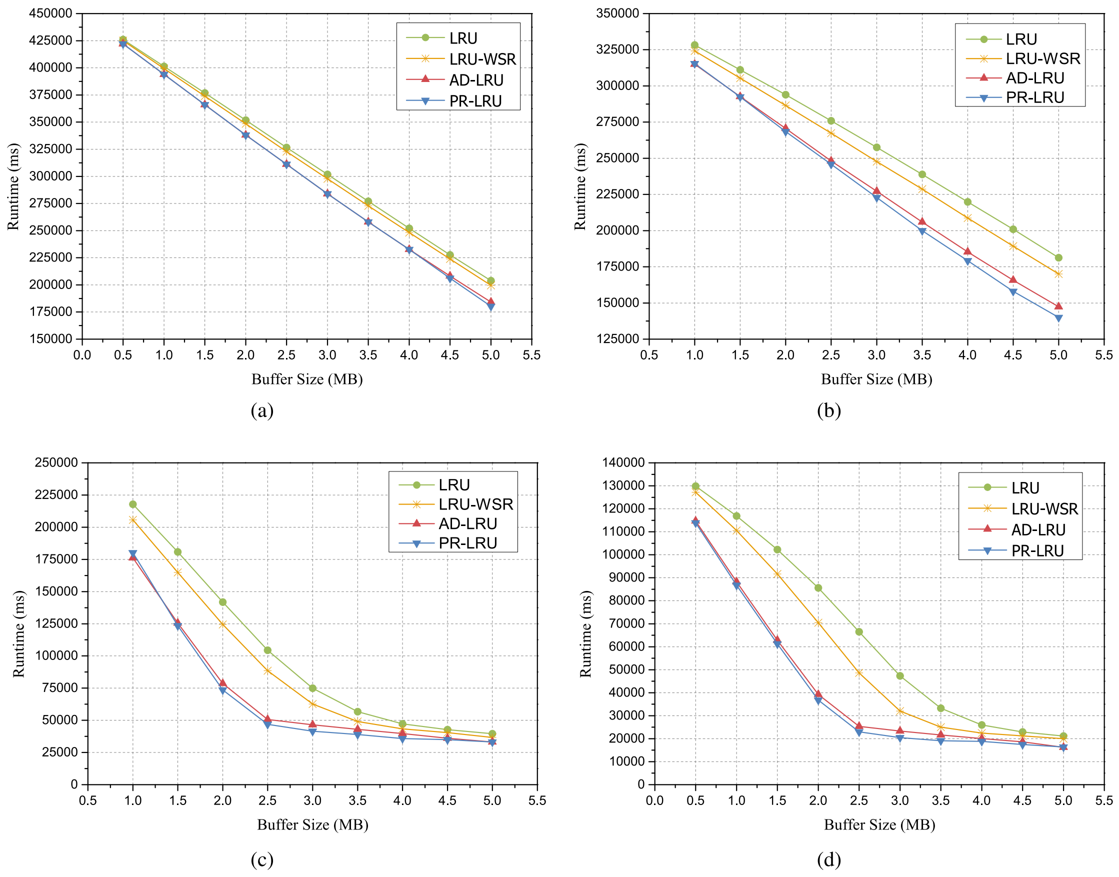

FIGURE 5. Comparison of the runtime on different traces. (a) T_1. (b) T_2. (c) T_3. (d) T_4.

four algorithms. If the buffer size is particularly small, the hit ratio is very low and the number of physical write operations is small. This occurs because the PR-LRU will evict clean pages in the victim LRU list first. When the PR-LRU selects a replacement page from the hot LRU list, it will calculate the probability of reference and select the minimum page as a replacement. Due to the PR strategy, the number of write operations for trace T_4 is reduced by 44.1%, 32.4%, and 7.6% compared with LRU [13], LRU-WSR [21] and AD-LRU [23].

Fig. 5 shows the runtime of the four algorithms for different buffer sizes. The PR-LRU algorithm has the lowest runtime. This differences in the number of flash memory write operations are not particularly large as the result of the asymmetric I/O latencies. Therefore, the general design of the buffer replacement algorithms is focused on decreasing the write count. When the buffer size is large, the runtime decreases slower than the memory time increases. As a consequence, PR-LRU reduces the runtime for trace T_4 by 36.7%, 28.3%, and 5.1%, respectively compared to LRU [13], LRU-WSR [21] and AD-LRU [23].

V. CONCLUSION AND FUTURE WORK

In this paper, a novel buffer replacement algorithm called PR-LRU is presented to enhance the performance of the flash memory and decrease the write count. We divide the buffer into a hot, cold, and victim LRU list. Pages from the hot or cold LRU list are inserted into the victim LRU list. The victim LRU list is preferred to replace clean pages than replace dirty pages. The PR-LRU provides an additional chance for pages to remain in the buffer. The probability of reference can be used to determine whether each page may be referenced. By using this method, we retain the pages that are more likely to be referenced in the future.

The proposed PR-LRU algorithm was tested on the flash simulation platform Flash-DBSim. The results show that the PR-LRU reduces the write count compared with other flash-based replacement buffer algorithms. Our algorithm provides up to 7% improvements for the hit ratio. Compared with other approaches, PR-LRU increases 36.7% in the overall runtime.

In future studies, we will attempt to adjust the lengths of three LRU lists dynamically to improve the performance of flash memory and enhance the flash memory's hit ratio.

ACKNOWLEDGMENT

The authors are grateful to the University of Science and Technology of China, Knowledge and Data Engineering Laboratory for providing the flash simulation platform Flash-DBSim.

REFERENCES

[1] S. A. Chamazcoti, Z. Delavari, S. Ghassem, and H. Asadi, "On endurance and performance of erasure codes in SSD-based storage systems," *Microelectron. Rel.*, vol. 55, no. 11, pp. 2453–2467, Aug. 2015.

[2] G. Jia, G. Han, and F. Wang, "Cost aware cache replacement policy in shared last-level cache for hybrid memory based fog computing," *Enterprise Inf. Syst.*, to be published, doi: 10.1080/17517575.2017.1295321.

[3] Y. Youwei, Y. Lamei, W. Yigang, H. Gengsheng, and C. Mei, "Sharing of larger medical DICOM imaging data-sets in cloud computing," *J. Med. Imag. Health Inf.*, vol. 5, no. 7, pp. 1390–1394, Nov. 2015.

[4] G. Jia, G. Han, J. Rodrigues, J. Lloret, and W. Li, "Coordinate memory deduplication and partition for improving performance in cloud computing," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2015.2511738.

[5] A. Dobri *et al.*, "Development and application of the oxide stress separation technique for the measurement of ONO leakage currents at low electric fields in 40 nm floating gate embedded-flash memory," *Microelectron. Rel.*, vol. 69, pp. 47–51, Dec. 2017.

[6] A. Fukami, S. Ghose, Y. Luo, Y. Cai, and O. Mutlu, "Improving the reliability of chip-off forensic analysis of NAND flash memory devices," *Digit. Investigat.*, vol. 20, pp. S1–S11, Jan. 2017.

[7] F. R. Ihmig, S. G. Shirley, and H. Zimmermann, "Batch screening of commercial serial flash-memory integrated circuits for low-temperature applications," *Cryogenics*, vol. 71, pp. 39–46, May 2015.

[8] Y. Kanza and H. Yaari, "External sorting on flash storage: Reducing cell wearing and increasing efficiency by avoiding intermediate writes," *VLDB J.*, vol. 25, no. 4, pp. 495–518, Mar. 2016.

[9] C. G. Kim, K. J. Kim, and J. Lee, "NAND flash memory system based on the Harvard buffer architecture for multimedia applications," *Multimedia Tools Appl.*, vol. 74, no. 16, pp. 6287–6302, Jan. 2014.

[10] K. M. Kobayashi, S. Miyazaki, and Y. Okabe, "Competitive buffer management for multi-queue switches in QoS networks using packet buffering algorithms," *Theor. Comput. Sci.*, vol. 675, pp. 27–42, Feb. 2017.

[11] G. Jia, G. Han, J. Jiang, and L. Liu, "Dynamic adaptive replacement policy in shared last-level cache of DRAM/PCM hybrid memory for big data storage," *IEEE Trans. Ind. Informat.*, to be published, doi: 10.1109/TII.2016.2645941.

[12] Y. Yuan, D. Chen, and L. Yan, "Interactive three-dimensional segmentation using region growing algorithms," *J. Algorithms Comput. Technol.*, vol. 9, no. 2, pp. 199–214, Aug. 2015.

[13] A. Panagakis, A. Vaios, and I. Stavrakakis, "Approximate analysis of LRU in the case of short term correlations," *Comput. Netw.*, vol. 52, no. 6, pp. 1142–1152, Jan. 2008.

[14] J. No, "NAND flash memory-based hybrid file system for high I/O performance," *J. Parallel Distrib. Comput.*, vol. 72, no. 12, pp. 1680–1695, Aug. 2012.

[15] D. Pan *et al.*, "Buffer management for streaming media transmission in hierarchical data of opportunistic networks," *Neurocomputing*, vol. 193, pp. 42–50, Feb. 2016.

[16] C. C. Sobin, "An efficient buffer management policy for DTN," *Procedia Comput. Sci.*, vol. 93, pp. 309–314, Sep. 2016.

[17] J. P. van Zandwijk, "A mathematical approach to NAND flash-memory descrambling and decoding," *Digit. Investigat.*, vol. 12, pp. 41–52, Feb. 2015.

[18] D. Wei *et al.*, "A page-granularity wear-leveling (PGWL) strategy for NAND flash memory-based sink nodes in wireless sensor networks," *J. Netw. Comput. Appl.*, vol. 63, pp. 125–139, Feb. 2016.

[19] P. Desnoyers, "Analytic modeling of SSD write performance," *ACM Trans. Storage*, vol. 10, no. 10, pp. 193–206, Jun. 2012.

[20] S. Y. Park *et al.*, "CFLRU: A replacement algorithm for flash memory," in *Proc. Int. Conf. Compil.*, Oct. 2006, vol. 55. no. 3, pp. 234–241.

[21] H. Jung, H. Shim, S. Park, S. Kang, and J. Cha, "LRU-WSR: Integration of LRU and writes sequence reordering for flash memory," *IEEE Trans. Consum. Electron.*, vol. 54, no. 3, pp. 1215–1223, Aug. 2008.

[22] Z. Li *et al.*, "CCF-LRU: A new buffer replacement algorithm for flash memory," *IEEE Trans. Consum. Electron.*, vol. 55, no. 3, pp. 1351–1359, Jun. 2009.

[23] P. Jin. Y. Ou. T. Härder, and Z. Li, "AD-LRU: An efficient buffer replacement algorithm for flash-based databases," *Data Knowl. Eng.*, vol. 72, pp. 83–102, Oct. 2012.

[24] R. Jin, H. J. Cho, and T. S. Chung, "LS-LRU: A lazy-split LRU buffer replacement policy for flash-based B+-tree index," *J. Inf. Sci. Eng.*, vol. 31, no. 3, pp. 1113–1132, Feb. 2015.

[25] G. Jia, G. Han, J. Jiang, and J. J. P. C. Rodrigues, "PARS: A scheduling of periodically active rank to optimize power efficiency for main memory," *J. Netw. Comput. Appl.*, vol. 58, pp. 327–336, Dec. 2015.

[26] M. Lin, S. Chen, G. Wang, and T. Wu, "HDC: An adaptive buffer replacement algorithm for NAND flash memory-based databases," *Opt. Int. J. Light Electron Opt.*, vol. 125, no. 3, pp. 1167–1173, Jul. 2014.

[27] S. H. Leong, A. Parodi, and D. Kranzlmüller, "A robust reliable energy-aware urgent computing resource allocation for flash-flood ensemble forecasting on HPC infrastructures for decision support," *Future Generat. Comput. Syst.*, vol. 68, pp. 136–149, Sep. 2017.

YOUWEI YUAN was born in 1966. He received the Ph.D. degree in computer science from the Wuhan University of Technology, China, in 2007. He is currently a Professor of Computer Science with Hangzhou Dianzi University, Hangzhou, China. He has authored over 50 technical papers in prestigious journals and conferences, 30 papers been indexed by SCI and EI. His research interests include artificial intelligence, data mining, cloud computing, big data analysis, and distributed parallel processing.

YETING SHEN was born in 1992. She is currently pursuing the Ph.D. degree in computer science from Hangzhou Dianzi University, Hangzhou, China. Her research interests include energy efficient hardware, devices, and designs for cloud computing platform.

WANQING LI received the B.S. and Ph.D. degrees from LanZhou University, LanZhou, China, in 2002 and 2007, respectively. Currently, he is an Associate Professor with the School of Computer Since and Technology, Hangzhou Dianzi University. His research interests are in data mining, driver behavior, and ITS. He achieved over ten projects supported by four traffic engineering corporations from 2007 to today.

DONGJIN YU is currently a Professor with Hangzhou Dianzi University, where he is the Dean of the Excellence Honors School and the Director of the Institute of Computer Software. His research efforts include big data, service computing, and program comprehension. He is especially interested in the novel approaches to constructing large enterprise information systems effectively and efficiently by emerging advanced information technologies. He is a member of the ACM, and a Senior Member of the China Computer Federation (CCF). He is also a member of the Technical Committee of Software Engineering CCF and the Technical Committee of Service Computing CCF.

LAMEI YAN was a Visiting Professor of RMIT University, Australia, from 2012 to 2013. She is currently a Professor with the School of Digital Media and Design. Her research interests include image processing, computer aided design, and finite element analysis.

YIFEI WANG was born in 1995. He is currently pursuing the bachelor's degree in automation, Hangzhou Dianzi University, Hangzhou, China. His research interests include intelligent control theory and application, and robust control.